1 String Matching

1.1 Rabin-Karp algorithm

Rabin-Karp string searching algorithm calculates a numerical (hash) value for the pattern \( p \), and for each \( m \)-character substring of text \( t \). Then it compares the numerical values instead of comparing the actual symbols. If any match is found, it compares the pattern with the substring by naive approach. Otherwise it shifts to next substring of \( t \) to compare with \( p \).

We can compute the numerical (hash) values using Horner’s rule.

Lets assume, \( h_0 = k \)

\[
h_1 = d (k - p[1].d^{m-1}) + p[m+1]
\]

Suppose, we have given a text \( t = [3, 1, 4, 1, 5, 2] \) and \( m = 5, q = 13; \)

\( t_0 = 31415 \)

So \( t_1 = 10(31415 - 10^5 1 + t[5+1]) \)

\[
= 10(31415 - 10^4 3) + 2
\]

\[
= 10(1415) + 2 = 14152
\]

Here \( p \) and substring \( t_i \) may be too large to work with conveniently. The simple solution is, we can compute \( p \) and the \( t_i \) modulo a suitable modulus \( q \).

So for each \( i \),

\[
h_{i+1} = (d (h_i - t[i+1].d^{m-1}) + t[m+i+1]) \mod q
\]

The modulus \( q \) is typically chosen as a prime such that \( d.q \) fits within one computer word.

**Algorithm**

Compute \( h_p \) (for pattern \( p \))

Compute \( h_t \) (for the first substring of \( t \) with \( m \) length)

For \( i = 1 \) to \( n - m \)

If \( h_p = h_t \)

Match \( t[i \ldots i + m] \) with \( p \), if matched return 1

Else

\[
h_t = (d (h_t - t[i+1].d^{m-1}) + t[m+i+1]) \mod q
\]

End

Suppose, \( t= 2359023141526739921 \) and \( p = 31415 \),

Now, \( h_p = 7 \) \((31415 = 7 \mod 13)\)

substring beginning at position 7 = valid match
This algorithm has a significant improvement in average-case running time over naive approach.